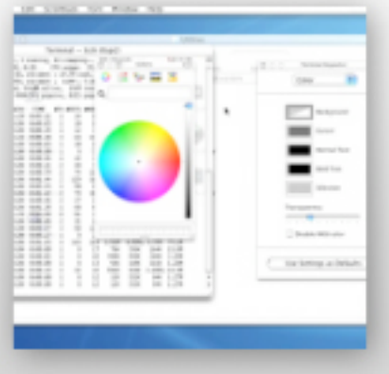
Messaging on Mac OS X

A Survey of Techniques

Session 106



Messaging on Mac OS X

A Survey of Techniques

Steve Zellers
Deptartment of Interspecies Communications
zellers@apple.com

Introduction

- Interapplication Communication
- Applications as building blocks
- Different frameworks; different capabilities
- Choose an appropriate framework

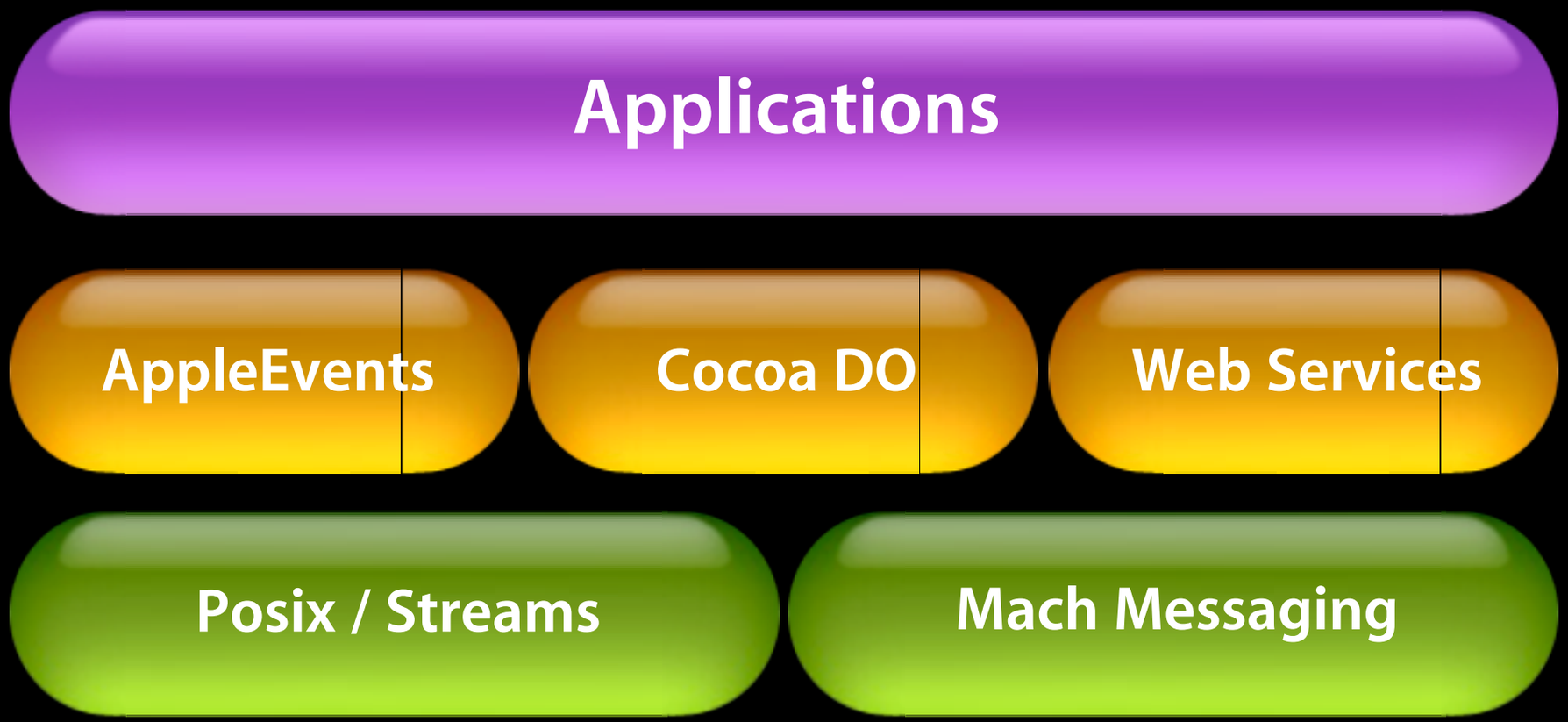
What You'll Learn

- Overview of available frameworks
- Questions to ask yourself
- Participating in an existing conversation
- Finding someone to talk to!

Technology Frameworks

- Unix and Mach
  - `pipe(2)`, `socket(2)`, `notify(3)`, `Shared Memory`, `mach_msg`.
- AppleEvents and ObjC DO
  - Local and remote
  - Scripting technologies
- Web Services

Architecture Diagram



UNIX Messaging

- Typically stream based
- `pipe(2)`—inherited or named pipes
- `socket(2)`—`AF_UNIX, AF_TCP`
- `select(2)`—block for events
- shared memory—`shm_open(2)`

UNIX Messaging: Pipes/Sockets

- Most portable
- Not well supported by the Mac OS X Application frameworks
  - `select(2)` emulation via `CFStream`
- Not efficient: linear time vs. message size
- Best use: reading output from a forked command

UNIX Messaging: Shared Memory

- Can be very efficient
- Usually: one writer, many readers
- 32-bit atomic updates
- Can require external synchronization
- Best use: sharing common data between processes

UNIX Messaging: notify(3)

- New for Panther
- Allows “cheap” “events”
- Events may be coalesced
- Delivery by polling, file descriptor or `mach_msg`
- A better `signal(3)` for Mac OS X

UNIX Messaging: notify(3)

```
#define kMyName "com.apple.notifytest"
{
  // “Client” registers for notifications
  int notify_fd, token;
  notify_register_file_descriptor(kMyName,&notify_fd, 0, &token);
```

UNIX Messaging: notify(3)

```
#define kMyName "com.apple.notifytest"
{
  // “Client” registers for notifications
  int notify_fd, token;
  notify_register_file_descriptor(kMyName,&notify_fd, 0, &token);

  // “Client” waits for the notification
  uint32_t val;
  int ctRead = read(notify_fd, &val, sizeof(val));
  if (ctRead > 0) { doSomethingAboutIt() }
  notify_cancel(token);
}
```

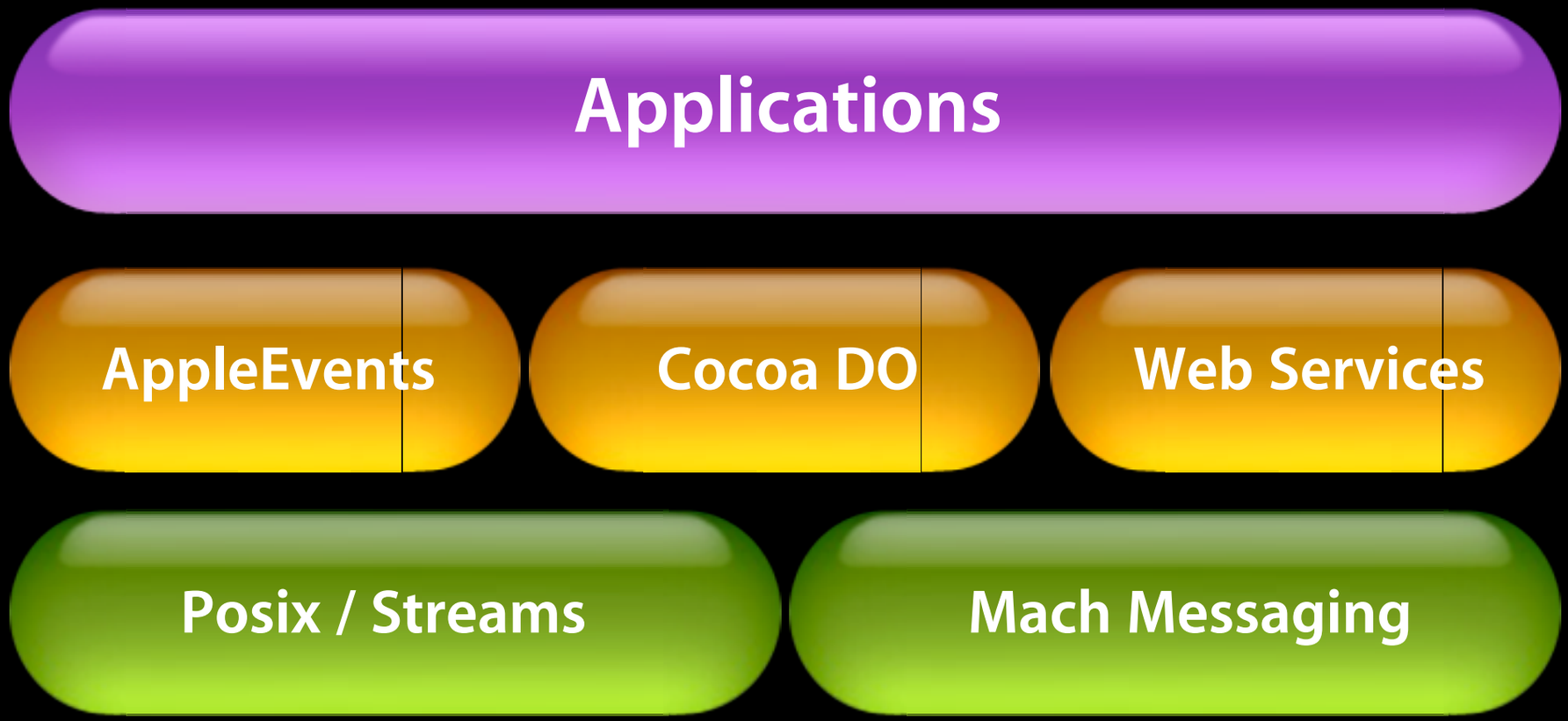
UNIX Messaging: notify(3)

```
#define kMyName "com.apple.notifytest"
{
  // “Client” registers for notifications
  int notify_fd, token;
  notify_register_file_descriptor(kMyName,&notify_fd, 0, &token);

  // “Client” waits for the notification
  uint32_t val;
  int ctRead = read(notify_fd, &val, sizeof(val));
  if (ctRead > 0) { doSomethingAboutIt() }
  notify_cancel(token);
}
```

```
{
  // “Server” sends notifications
   notify_post("com.apple.notifytest");
}
```

Architecture Diagram



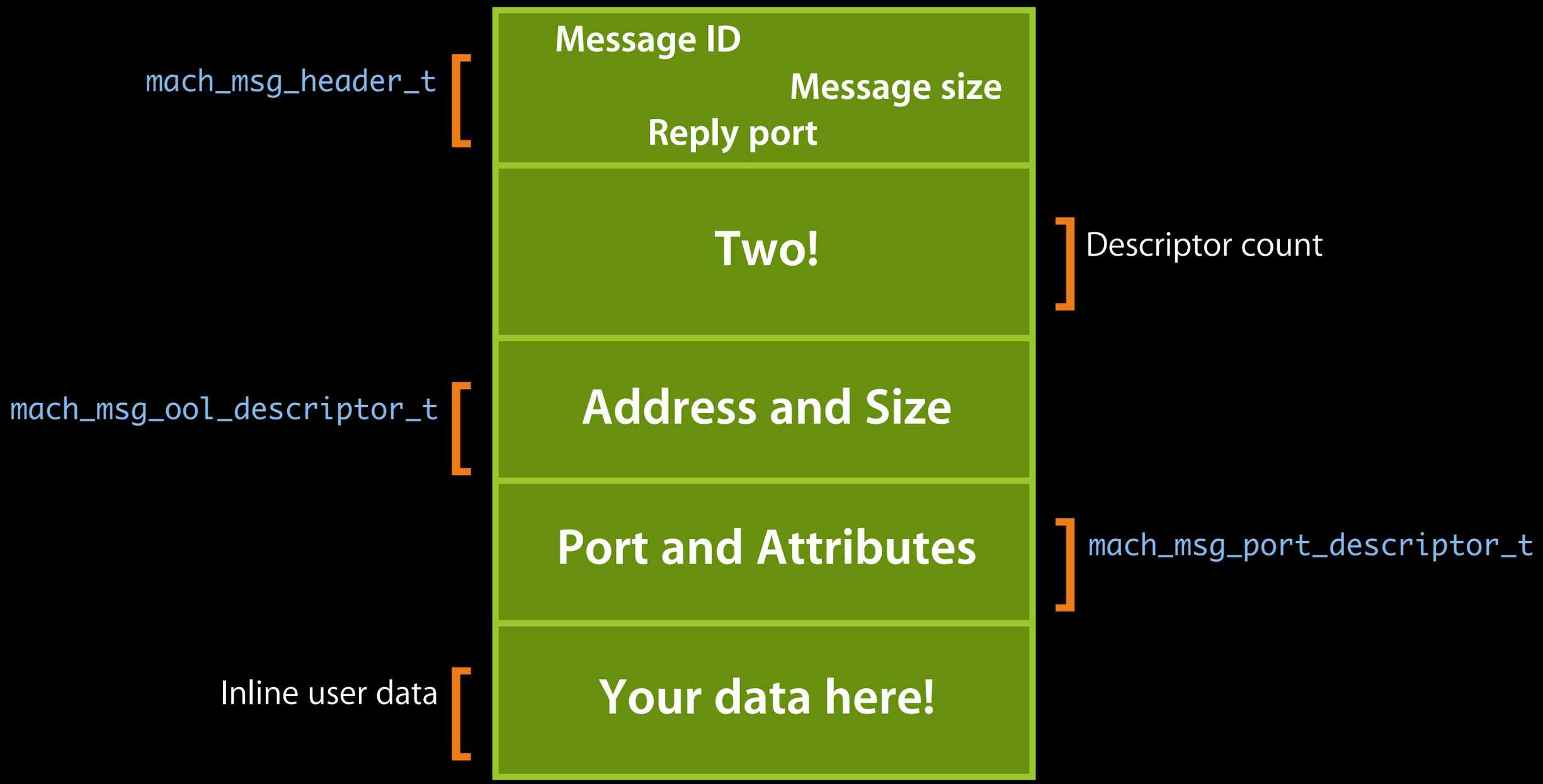
Mach Messaging: Concepts

- Servers
  - In the kernel
  - Other processes
- Terms to know:
  - `mach_msg`
  - `mach_msg_header_t`
  - `mach_port_t`
  - Out of band memory

Mach: Memory Descriptors

- Virtual memory copies
- Read only or copy on write
- Page align the memory!
- Plan your memory management
  - `vm_deallocate()`

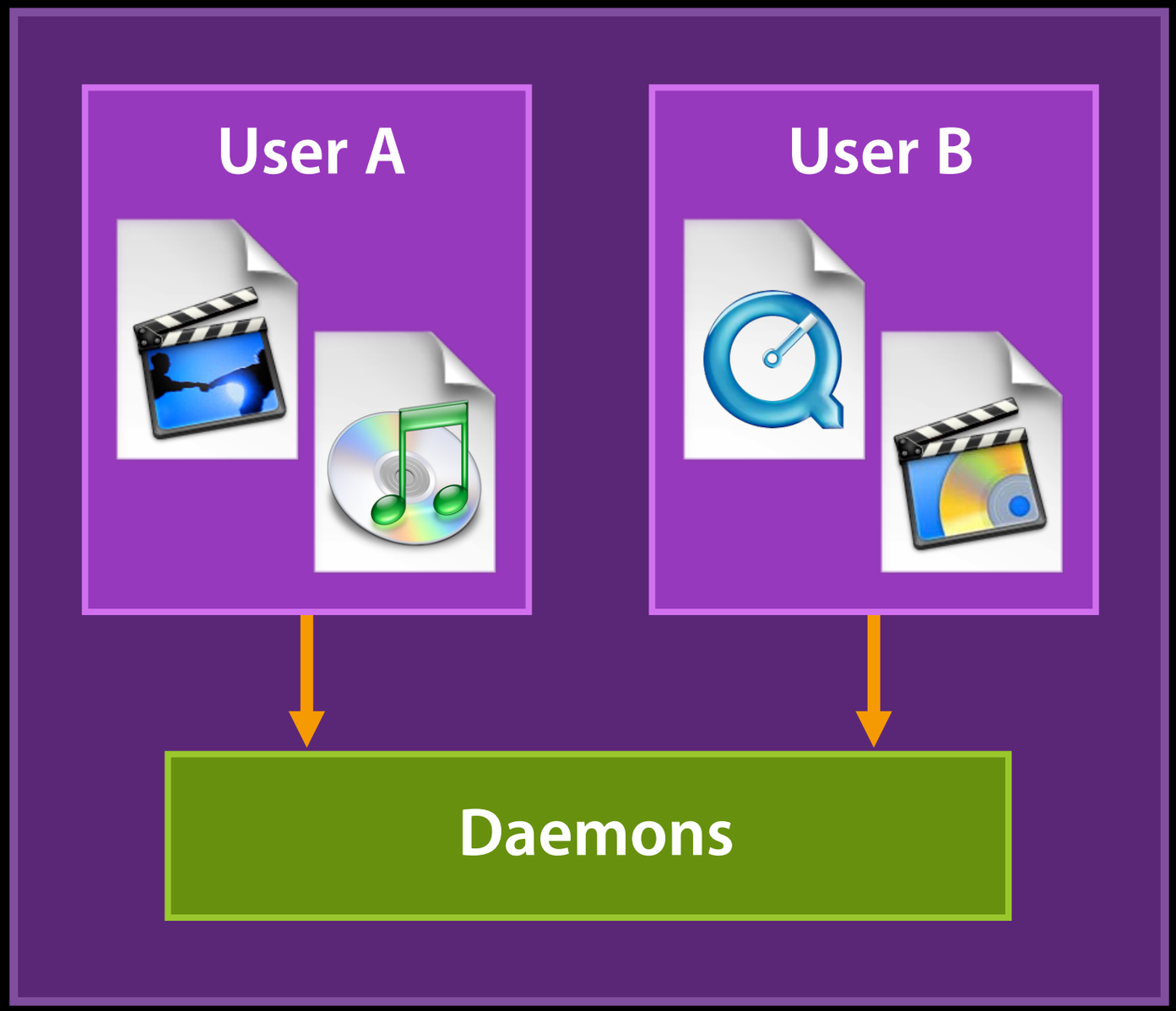
Mach: Message Structure



Mach: bootstrap server

- Discovery of mach services
- Associates ports with a (well known) name
- Bootstrap servers layer/nest:
  - Primordial bootstrap server: daemons
  - Per session: applications
- Security: Restricts visible ports

Mach: Bootstrap Server



Mach: Interface Generator (MIG)

- An IDL for describing mach messages
- Used to implement servers
- Declares sets of messages with unique IDs
- Generates server stubs and client glue

Mach: MIG Example

```
routine Service_Checkin(
        serverPort      : mach_port_t;
 in     id              : c_string[128];
 out    listenPort      : mach_port_make_send_t);
```

Mach: MIG Example

```
routine Service_Checkin(
        serverPort      : mach_port_t;
 in     id              : c_string[128];
 out    listenPort      : mach_port_make_send_t);
```

```
Client glue: (Example.h, Example.c)
kern_return_t Service_Checkin(mach_port_t serverPort,
                              char* id,
                              mach_port_t* clientPort);
```

Mach: MIG Example

```
routine Service_Checkin(
        serverPort      : mach_port_t;
 in     id              : c_string[128];
 out    listenPort      : mach_port_make_send_t);
```

```
Client glue: (Example.h, Example.c)
kern_return_t Service_Checkin(mach_port_t serverPort,
                              char* id,
                              mach_port_t* clientPort);
```

```
Server glue: (ExampleServer.h)
kern_return_t _srv_Service_Checkin(mach_port_t serverPort,
                                   char* id,
                                   mach_port_t* clientPort);
PLUS: _src_server_routine(): for demux purposes
```

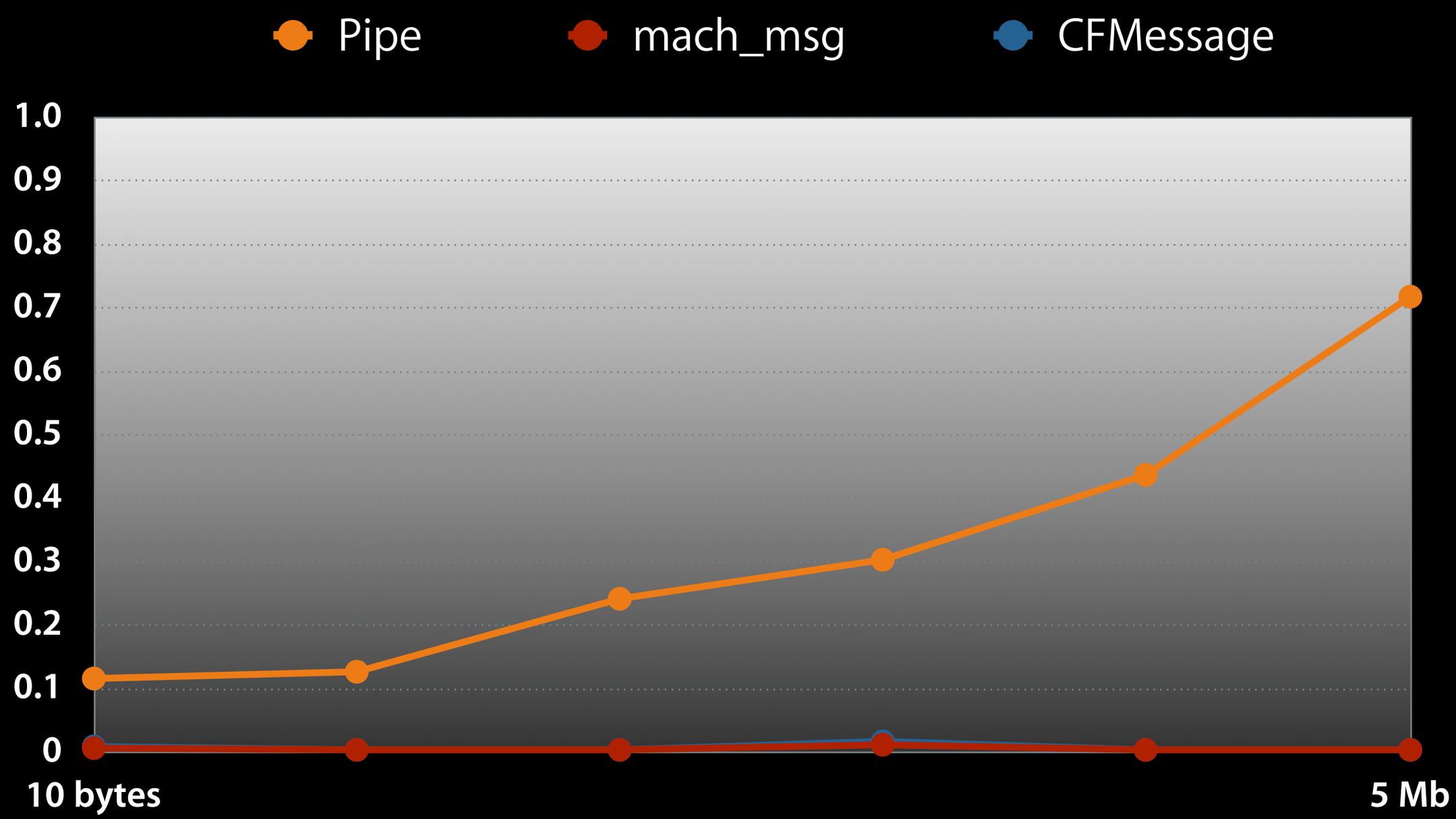
Mach: Handling Messages

- Demux based on `header.msgh_id`
- Process the message and return reply
- `mach_msg_server()`, `mach_msg_server_once()`
- Sending replies: roll your own
  - "Use the source, Luke (Darwin!)"

Mach: CoreFoundation support

- `CFMachPort`
  - Callback mechanism
    - `CFMachPortRunLoopSource`
  - Invalidation mechanism
  - No support for MIG demux or replies
- `CFMessagePort`
  - Leverages `CFMachPort` and bootstrap server
  - Cannot be used inter-session!
  - Highly recommended!

Mach: Performance



AppleEvents: Application IPC

- AppleEvents vs. `AppleEvent` vs. Apple “Events”
- Introduced in System 7
- Under the ApplicationServices umbrella
- Programmed behaviors and notifications
- Stateless
- Allows for scripting

AppleEvents: Suitability

- Between applications
- Discovery:
  - Application signature, process serial number, pid
  - New in Panther: `typeApplicationBundleID`
  - Jaguar restrictions:
    - Must be in the same bootstrap session
  - Panther enhancements:
    - UID based restrictions (daemons!)

AppleEvents: In Frameworks

- Carbon **AND** Cocoa
- Events are received from the RunLoop
- Usually synchronous, but can be suspended
- Scripting not free: `aete` resource
- Cocoa is almost free: `.scriptsuite` file

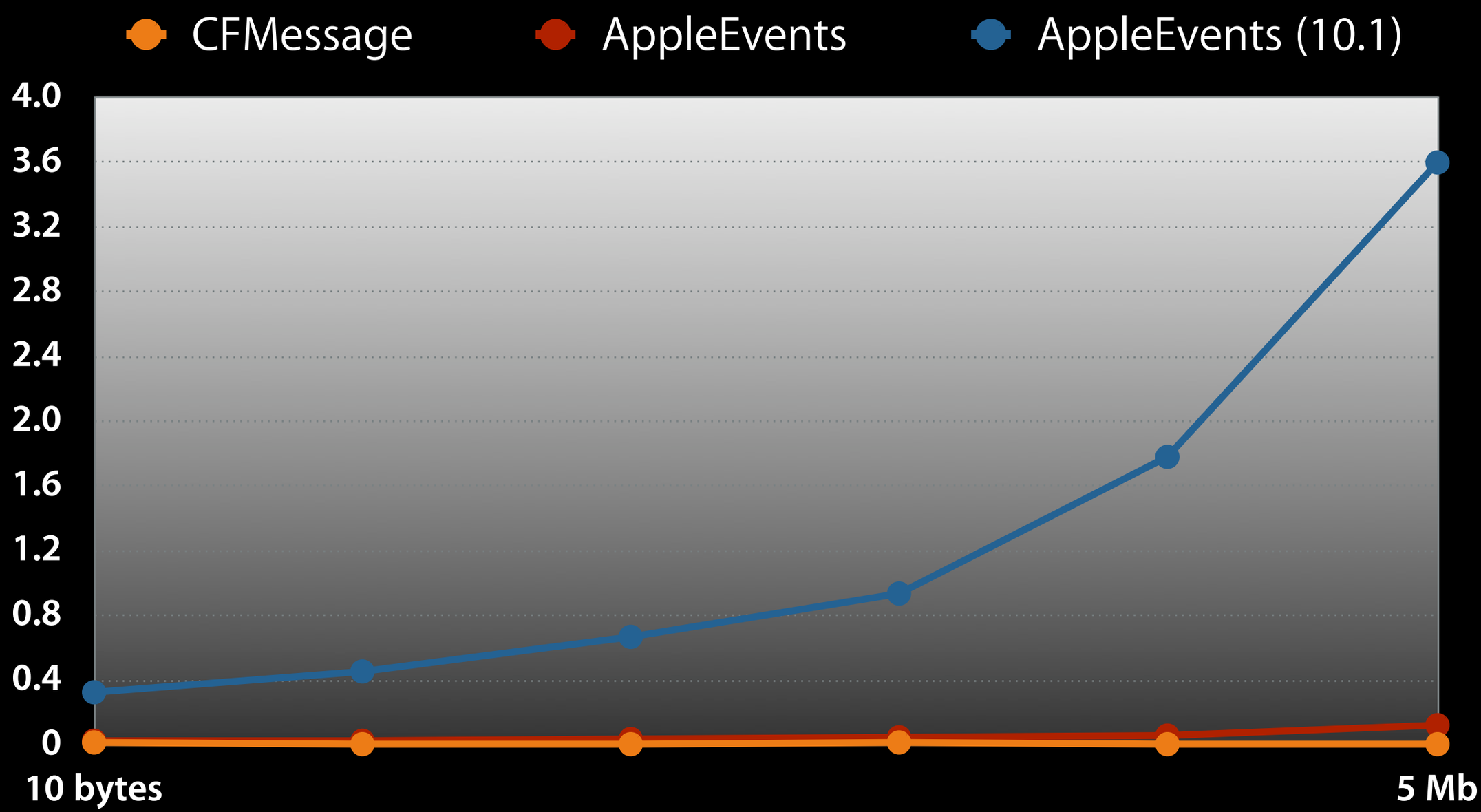
AppleEvents: Structure

- Simple containment hierarchy
- Typed Data, Lists, Records
- Data is opaque! Use accessors!
- An AppleEvent is just an AERecord with attributes.
  - event class and id; return id
- Data is “copied” when messaged

AppleEvents: Remote AppleEvents

- `typeApplicationURL`
  - `eppc://[user[:pw]@]host/App[?[uid=xxx][&pid=yyy]]`
- Muti-session additions: `uid` and `pid`
- New: `AEListRemoteProcesses` API
- Remote AE servers advertise on Rendezvous
  - `"_eppc._tcp."`

AppleEvents: Performance



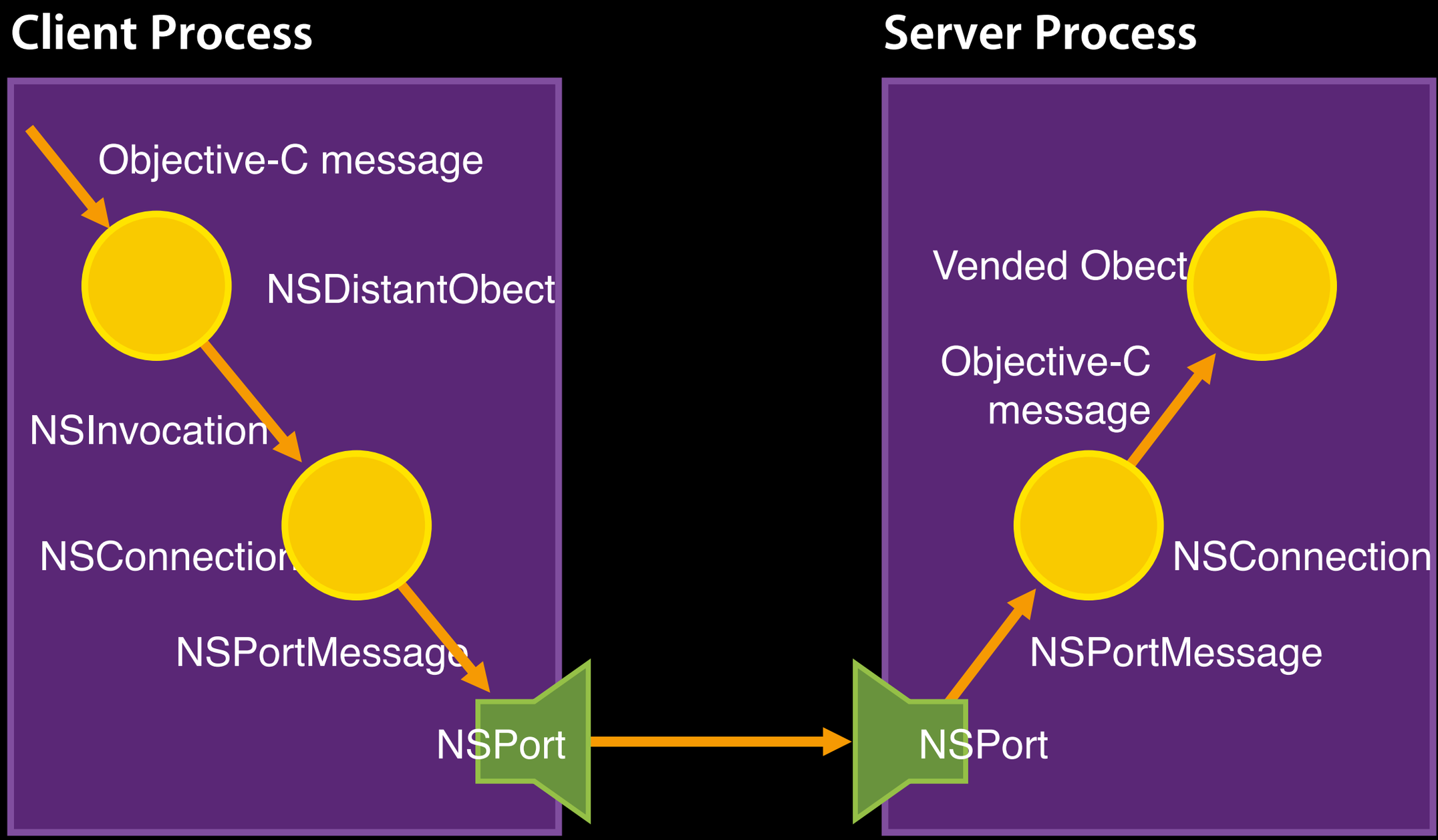
Distributed Objects: Cocoa

- Implement a “protocol”
- Services for other applications
- Object Proxies
  - Looks like an object!
- Inter process: mach messages
- Inter machine: sockets; discovery via Rendezvous

Distributed Objects: Objects

- `NSPort`: manages the serialization, transport
- `NSConnection`: object discovery and management
  - Bound to runloop
- `NSProxy` protocol:
  - `NSDistantObject`: client
  - `NSProtocolChecker`: server

Distributed Objects: Flow



Distributed Objects: Summary

- “Vend” ObjC objects
- NSConnections: set the timeout
- As efficient as possible…

  … but not as what you could come up with!

Web Services: WebServicesCore

- Uses HTTP[S] to to message XML conversations
- XML-RPC and SOAP 1.1 supported
- CoreFoundation and ObjC types
- AppleEvents/AppleScript
- New for Panther: Message Parsing API

Web Services: Sending a Request

- `WSMethodInvocationRef`
  - Schema
  - URL
- SetParameters
  - Order and values
- Invoke!
  - Synchronous and Asynchronous
- Extract values from reply

Web Services: Writing a Server

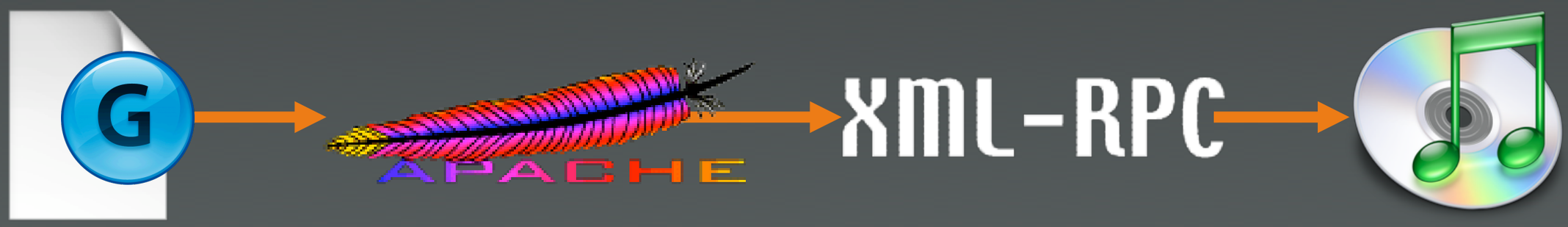
- `WSProtocolHandlerRef`
  - Schema
- `CopyRequestDictionary`
  - `CFData XML Document -> CFDictionary`
- Extract parameters and forumulate reply
- `CreateReplyDocument -> CFData XML Document`
- Send reply document
- Profit!

Web Services: Discovery

- No free lunch
- Textual Descriptions
  - Most common for XML-RPC
- WSDL can automate the process of describing web services
  - WSMakeStubs tool: C++, ObjC, Applescript

Demo

XML-RPC Apache Module Using New Server API



IPC Summary

- Local messaging:
  - pipe/socket: for legacy code
  - notify: broadcast state changes
  - mach_msg: complex servers
  - CFMessagePort: app to app
  - AppleEvents: whenever possible!
  - ObjC DO: New applications and services

Wrap Up

Roadmap

618 Building Applications for Managed Network Environments	**Pacific Heights** **Thu., 9:00am**
111 Writing Threaded Applications on Mac OS X	**Pacific Heights** **Thu., 2:00pm**
112 CFNetwork In Depth Using CFNetwork to Write Networked Applications	**Pacific Heights** **Fri., 5:00pm**
Stump the Experts Match Wits With Apple Engineering	**Presidio** **Wed., 6:30pm**

Who to Contact

Xavier Legros
Apple Worldwide Developer Relations
xavier@apple.com

Jason Yeo
Mac OS Technology Manager
Apple Worldwide Developer Relations
jason@apple.com

http://developer.apple.com/wwdc2003/urls.html

Reference Library

Documentation

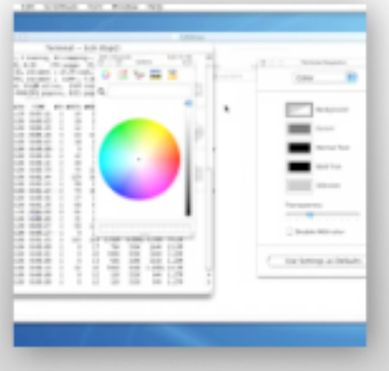
Kernighan and Pike: The Unix Programming Environment
http://cm.bell-labs.com/cm/cs/upe/

CMU Mach Kernel Documentation
ftp://ftp.cs.cmu.edu//afs/cs/project/mach/public/doc/unpublished

Inside Macintosh: IAC
http://developer.apple.com/techpubs/mac/IAC/IAC-2.html

WebServices Documentation
http://developer.apple.com/techpubs/macosx/Networking/WebServices

Q&A



http://developer.apple.com/wwdc2003/urls.html

WWDC 2003

Worldwide Developers Conference